TRON

Advanced Decentralized Blockchain Platform

Whitepaper Version: 2.0

TRON Protocol Version: 3.2

TRON Foundation

December 10th, 2018, San Francisco

1. Introduction

1.1 Vision

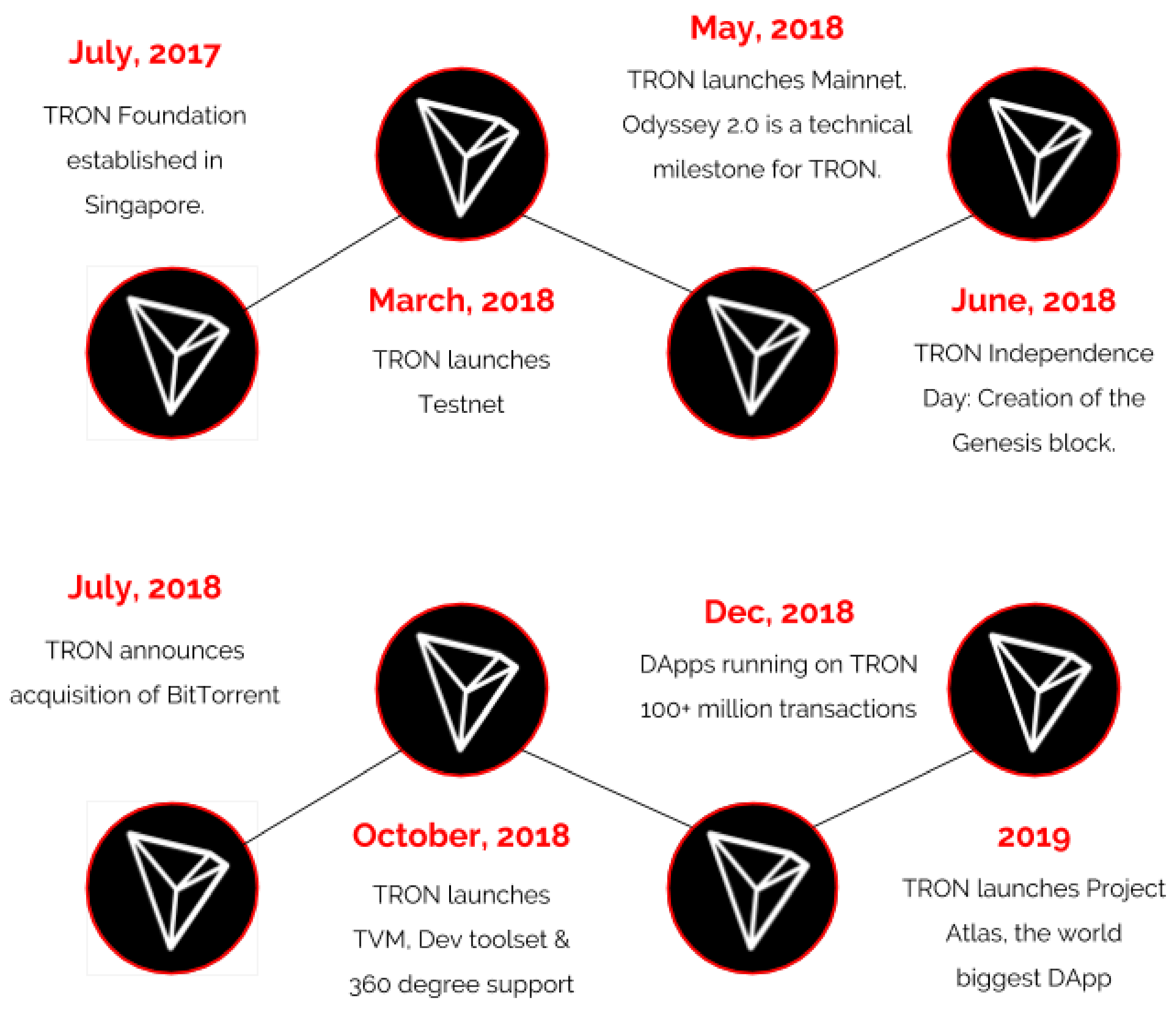
TRON is an ambitious project dedicated to the establishment of a truly decentralized Internet and its infrastructure. The TRON Protocol, one of the largest blockchain-based operating systems in the world, offers public blockchain support of high throughput, high scalability, and high availability for all Decentralized Applications (DApps) in the TRON ecosystem. The July 2018 acquisition of BitTorrent further cemented TRON's leadership in pursuing a decentralized ecosystem.

1.2 Background

The introduction of Bitcoin in 2009 revolutionized society's perception of the traditional financial system in the wake of the Great Recession (2007-2008). As centralized hedge funds and banks collapsed from speculation in opaque financial derivatives, blockchain technology provided a transparent universal ledger from which anybody could glean transaction information. The transactions were cryptographically secured using a Proof of Work (PoW) consensus mechanism, thus preventing double spend issues.

In late 2013, the Ethereum white paper proposed a network in which smart contracts and a Turing-complete Ethereum Virtual Machine (EVM) would allow developers to interact with the network through DApps. However, as transaction volumes in Bitcoin and Ethereum peaked in 2017, it was apparent from the low transaction throughput times and high transaction fees that cryptocurrencies like Bitcoin and Ethereum in their existing state were not scalable for widespread adoption. Thus, TRON was founded and envisioned as an innovative solution to these pressing scalability challenges.

1.3 History


July, 2017
TRON Foundation established in Singapore.
March, 2018
TRON launches Testnet
May, 2018
TRON launches Mainnet. Odyssey 2.0 is a technical milestone for TRON.
June, 2018
TRON Independence Day: Creation of the Genesis block.
July, 2018
TRON announces acquisition of BitTorrent
October, 2018
TRON launches TVM, Dev toolset & 360 degree support
Dec, 2018
DApps running on TRON 100+ million transactions
2019
TRON launches Project Atlas, the world biggest DApp

The TRON Foundation was established in July 2017 in Singapore. In December 2017, TRON had launched its open source protocol. The Testnet, Blockchain Explorer, and Web Wallet were all launched by March 2018. TRON Mainnet launched shortly afterward in May 2018, marking the Odyssey 2.0 release as a technical milestone. In June 2018, TRON declared its independence with the creation of the Genesis block, along with the July 2018 acquisition of BitTorrent. In October 2018, TRON launched the TRON Virtual Machine (TVM), a complete developers' toolset, and 360 support system. The TRON roadmap involves combining BitTorrent's 100 million users with the TRON network via Project Atlas, as well as fostering the developer community to launch exciting new DApps on the TRON network[1].

[1] V1.0 is available at https://tron.network/static/doc/white_paper_v_1_0.pdf

1.4 Terminology

Address/Wallet
An address or wallet consisting of account credentials on the TRON network are generated by a key pair, which consists of a private key and a public key, the latter being derived from the former through an algorithm. The public key is usually used for session key encryption, signature verification, and encrypting data that could be decrypted by a corresponding private key.

ABI
An application binary interface (ABI) is an interface between two binary program modules; usually one of these modules is a library or an operating system facility, and the other is a user run program.

API
An application programming interface (API) is mainly used for user clients development. With API support, token issuance platforms can also be designed by developers themselves.

Asset
In TRON's documents, asset is the same as token, which is also denoted as TRC-10 token.

Bandwidth Points (BP)
To keep the network operating smoothly, TRON network transactions use BP as fuel. Each account gets 5000 free daily BP and more can be obtained by freezing TRX for BP. Both TRX and TRC-10 token transfers are normal transactions costing BP. Smart contract deployment and execution transactions consume both BP and Energy.

Block
Blocks contain the digital records of transactions. A complete block consists of the magic number, block size, block header, transaction counter, and transaction data.

Block Reward
Block production rewards are sent to a sub-account (address/wallet). Super Representatives can claim their rewards on Tronscan or through the API directly.

Block Header
A block header is part of a block. TRON block headers contain the previous block’s hash, the Merkle root, timestamp, version, and witness address.

Cold Wallet
Cold wallet, also known as offline wallet, keeps the private key completely disconnected from any network. Cold wallets are usually installed on "cold" devices (e.g. computers or mobile phones staying offline) to ensure the security of TRX private key.

DApp
Decentralized Application is an App that operates without a centrally trusted party. An application that enables direct interaction/agreements/communication between end users and/or resources without a middleman.

gRPC
gRPC[2] (gRPC Remote Procedure Calls) is an open source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. Most common usage scenarios include connecting services in microservices style architecture and connecting mobile devices, and browser clients to backend services.

Hot Wallet
Hot wallet, also known as online wallet, allows user's private key to be used online, thus it could be susceptible to potential vulnerabilities or interception by malicious actors.

JDK
Java Development Kit is the Java SDK used for Java applications. It is the core of Java development, comprising the Java application environment (JVM+Java class library) and Java tools.

KhaosDB
TRON has a KhaosDB in the full-node memory that can store all the newly-forked chains generated within a certain period of time and supports witnesses to switch from their own active chain swiftly into a new main chain. See 2.2.2 State Storage for more details.

LevelDB
LevelDB was initially adopted with the primary goal to meet the requirements of fast R/W and rapid development. After launching the Mainnet, TRON upgraded its database to an entirely customized one catered to its very own needs. See 2.2.1 Blockchain Storage for more details.

Merkle Root
A Merkle root is the hash of all hashes of all transactions included as part of a block in a blockchain network. See 3.1 Delegated Proof of Stake (DPoS) for more details.

[2] https://en.wikipedia.org/wiki/GRPC

Public Testnet (Shasta)
A version of the network running in a single-node configuration. Developers can connect and test features without worrying about the economic loss. Testnet tokens have no value and anyone can request more from the public faucet.

RPC[3]
In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

Scalability
Scalability is a feature of the TRON Protocol. It is the capability of a system, network, or process to handle a growing amount of work or its potential to be enlarged to accommodate that growth.

SUN
SUN replaced drop as the smallest unit of TRX. 1 TRX = 1,000,000 SUN.

Throughput
High throughput is a feature of TRON Mainnet. It is measured in Transactions Per Second (TPS), namely the maximum transaction capacity in one second.

Timestamp
The approximate time of block production is recorded as Unix timestamp, which is the number of milliseconds that have elapsed since 00:00:00 01 Jan 1970 UTC.

TKC
Token configuration.

TRC-10
A standard of crypto token on TRON platform. Certain rules and interfaces are required to follow when holding an initial coin offering on TRON blockchain.

TRX
TRX stands for Tronix, which is the official cryptocurrency of TRON.

[3] https://en.wikipedia.org/wiki/Remote_procedure_call

2. Architecture

TRON adopts a 3-layer architecture divided into Storage Layer, Core Layer, and Application Layer. The TRON protocol adheres to Google Protobuf, which intrinsically supports multi-language extension.

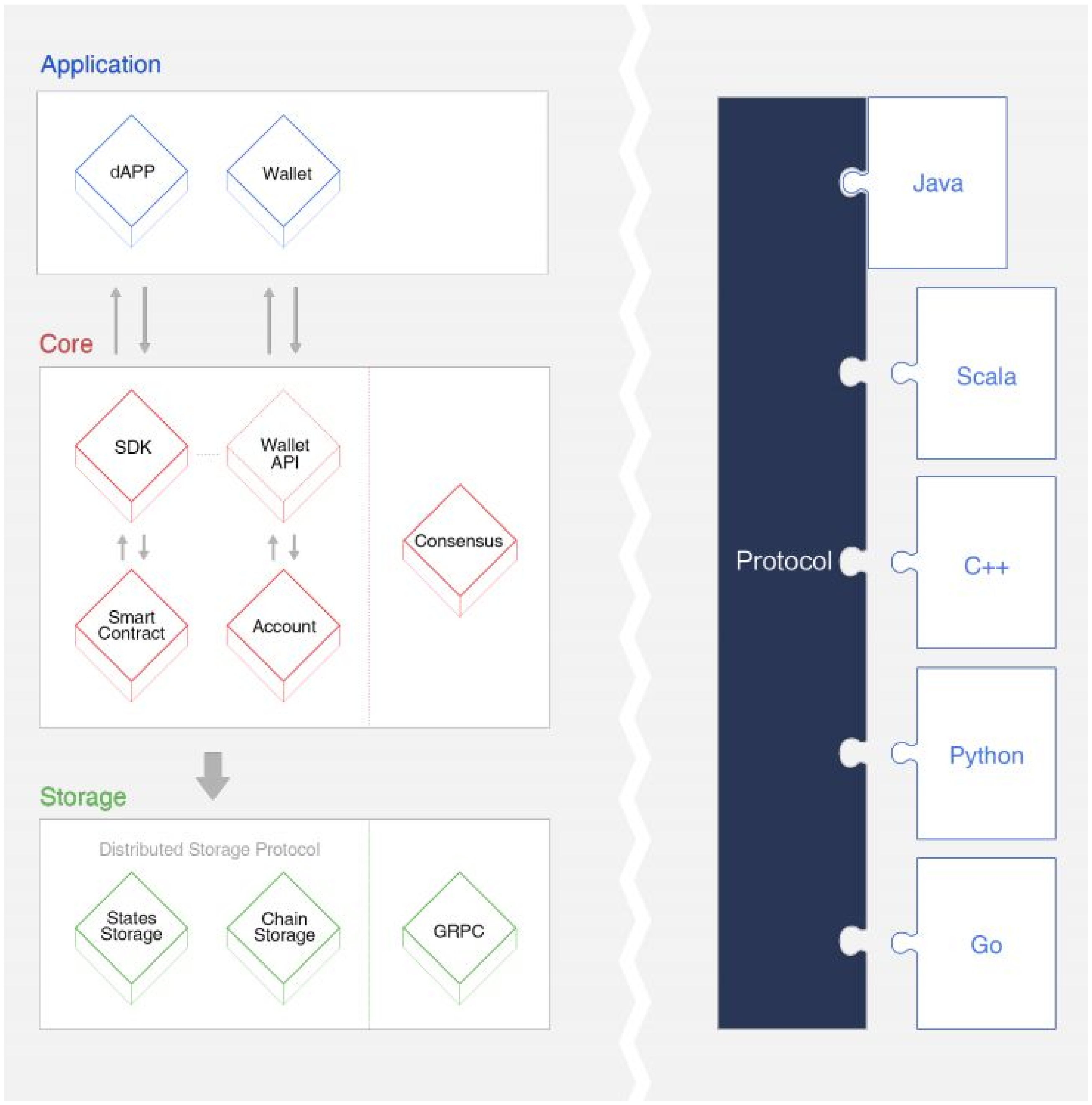


Figure 1: TRON 3-layer Architecture

2.1 Core

There are several modules in the core layer, including smart contracts, account management, and consensus. A stack-based virtual machine is implemented on TRON and an optimized instruction set is used. In order to better support DApp developers, Solidity[4] was chosen as the smart contract language, followed by future support of other advanced languages. In addition, TRON's consensus mechanism is based on Delegated Proof of Stake (DPoS) and many innovations were made in order to meet its unique requirements.

2.2 Storage

TRON designed a unique distributed storage protocol consisting of Block Storage and State Storage. The notion of a graph database was introduced into the design of the storage layer to better meet the need for diversified data storage in the real world.

2.2.1 Blockchain Storage

TRON blockchain storage chooses to use LevelDB, which is developed by Google and proven successful with many companies and projects. It has high performance and supports arbitrary byte arrays as both keys and values, singular get, put and delete, batched put and delete, bi-directional iterators, and simple compression using the very fast Snappy algorithm.

2.2.2 State Storage

TRON has a KhaosDB in the full-node memory that can store all the newly forked chains generated within a certain period of time and supports witnesses to switch from their own active chain swiftly into a new main chain. It can also protect blockchain storage by making it more stable from being terminating abnormally in an intermediate state.

2.3 Application

Developers can create a diverse range of DApps and customized wallets on TRON. Since TRON enables smart contracts to be deployed and executed, the opportunities of utility applications are unlimited.

[4] Solidity official documentation: https://solidity.readthedocs.io/

2.4 Protocol

TRON protocol adheres to Google Protocol Buffers[5], which is a language-neutral, platform-neutral, and extensible way of serializing structured data for use in communications protocols, data storage, and more.

2.4.1 Protocol Buffers

Protocol Buffers (Protobuf) is a flexible, efficient, automated mechanism for serializing structured data, similar to JSON or XML, but much smaller, faster and simpler.

Protobuf (.proto) definitions can be used to generate code for C++, Java, C#, Python, Ruby, Golang, and Objective-C languages through the official code generators. Various third-party implementations are also available for many other languages. Protobuf eases development for clients by unifying the API definitions and also optimizing data transfers. Clients can take the API .proto from TRON's protocol repository and integrate through the automatically-generated code libraries.

As a comparison, Protocol Buffers is 3 to 10 times smaller and 20 to 100 times faster than XML, with less ambiguous syntax. Protobuf generates data access classes that are easier to use programmatically.

2.4.2 HTTP

TRON Protocol provides a RESTful HTTP API alternative to the Protobuf API. They share the same interface but the HTTP API can be readily used in javascript clients.

2.5 TRON Virtual Machine (TVM)

The TVM is a lightweight, Turing complete virtual machine developed for TRON's ecosystem. The TVM connects seamlessly with the existing development ecosystem to provide millions of global developers with a custom-built blockchain system that is efficient, convenient, stable, secure, and scalable.

2.6 Decentralized Exchange (DEX)

[5] Google Protocol Buffers official documentation: https://developers.google.com/protocol-buffers/

The TRON network natively supports decentralized exchange functions. A decentralized exchange consists of multiple trading pairs. A trading pair (notation “Exchange”) is an Exchange Market between TRC-10 tokens, or between a TRC-10 token and TRX. Any account can create a trading pair between any tokens, even if the same pair already exists on the TRON network. Trading and price fluctuations of the trading pairs follow the Bancor Protocol[6]. The TRON network stipulates that the weights of the two tokens in all trading pairs are equal, so the ratio of their balances is the price between them. For example, consider a trading pair containing two tokens, ABC and DEF. ABC has a balance of 10 million and DEF has a balance of 1 million. Since their weights are equal, 10 ABC = 1 DEF. This means that the ratio of ABC to DEF is 10 ABC per DEF.

2.7 Implementation

The TRON blockchain code is implemented in Java and was originally a fork from EthereumJ.

[6] Bancor Protocol official website: https://about.bancor.network/protocol/

3. Consensus

3.1 Delegated Proof of Stake (DPoS)

The earliest consensus mechanism is the Proof of Work (PoW) consensus mechanism. This protocol is currently implemented in Bitcoin[7] and Ethereum[8]. In PoW systems, transactions broadcast through the network are grouped together into nascent blocks for miner confirmation. The confirmation process involves hashing transactions using cryptographic hashing algorithms until a merkle root has been reached, creating a merkle tree:

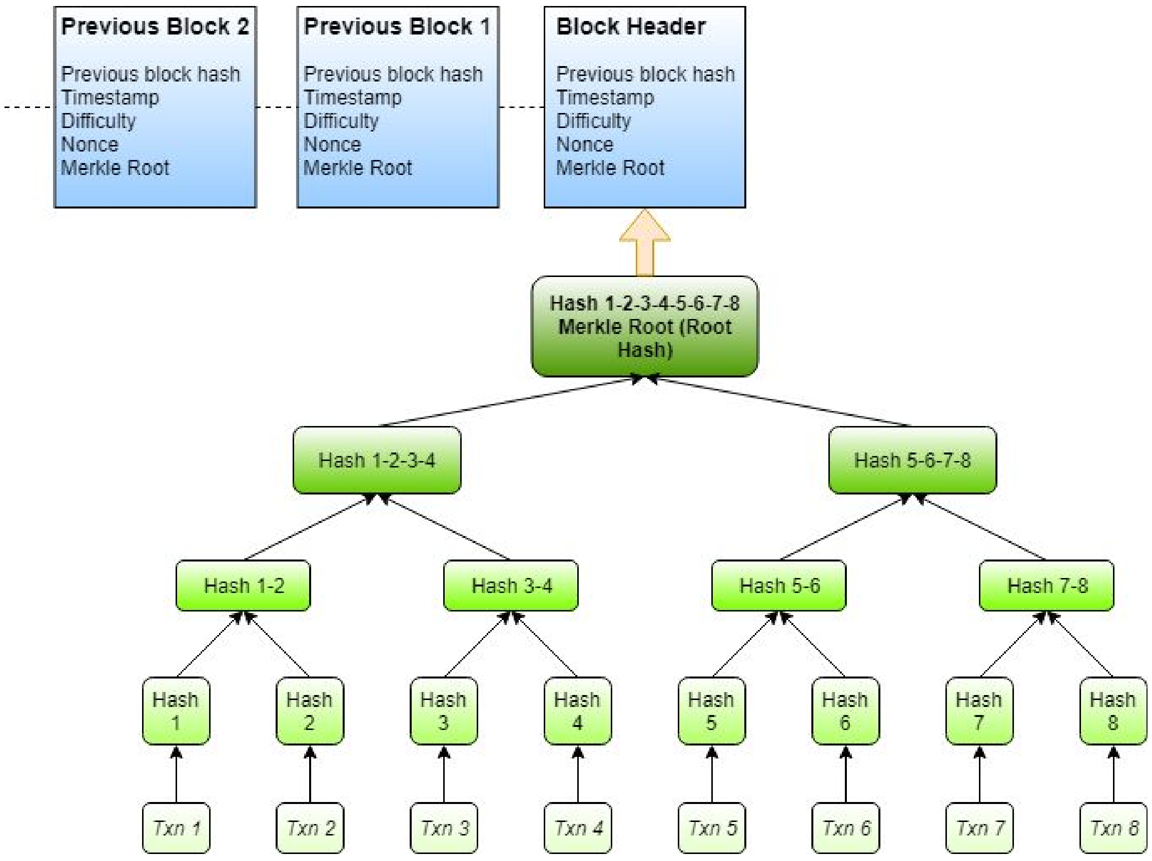


Figure 2: 8 TRX transactions are hashed into the merkle root. This merkle root is then included in the block header, which is attached to the previously confirmed blocks to form a blockchain. This allows for easy and transparent tracking of transactions, timestamps, and other related information.

[7] Bitcoin whitepaper: https://bitcoin.org/bitcoin.pdf
[8] Ethereum whitepaper: https://github.com/ethereum/wiki/wiki/White-Paper

Cryptographic hashing algorithms are useful in network attack prevention because they possess several properties[9]:

- ***Input/Output length size*** - The algorithm can pass in an input of any length in size, and outputs a fixed length hash value.
- ***Efficiency*** - The algorithm is relatively easy and fast to compute.
- ***Preimage resistance*** - For a given output *z*, it is impossible to find any input *x* such that $h(x) = z$. In other words, the hashing algorithm $h(x)$ is a one-way function in which only the output can be found, given an input. The reverse is not possible.
- ***Collision resistance*** - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$. In other words, the probability of finding two different inputs hashing to the same output is extremely low. This property also implies *second preimage resistance*.
- ***Second preimage resistance*** - Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$. While this property is similar to *collision resistance*, the property differs in that it is saying an attacker with a given x_1 will find it computationally infeasible to find any x_2 hashing to the same output.
- ***Deterministic*** - maps each input to one and only one output.
- ***Avalanche effect*** - a small change in the input results in an entirely different output.

These properties give the cryptocurrency network its intrinsic value by ensuring attacks do not compromise the network. When miners confirm a block, they are rewarded tokens as a built-in incentive for network participation. However, as the global cryptocurrency market capitalization steadily increased, the miners became centralized and focused their computing resources on hoarding tokens as assets, rather than for network participation purposes. CPU miners gave way to GPUs, which in turn gave way to powerful ASICs. In one notable study, the total power consumption of Bitcoin mining has been estimated to be as high as 3 GW[10], comparable to Ireland's power consumption. This same study projected total power consumption to reach 8 GW in the near future.

To solve the energy waste issue, the Proof of Stake (PoS) consensus mechanism was proposed by many new networks. In PoS networks, token holders lock their token balances to become block validators. The validators take turns proposing and voting on the next block. However, the problem with standard PoS is that validator influence correlates directly to the amount of tokens locked up. This results in parties hoarding large amounts of the network's base currency wielding undue influence in the network ecosystem.

The TRON consensus mechanism uses an innovative Delegated Proof of Stake system in which 27 Super Representatives (SRs) produce blocks for the network. Every 6 hours, TRX account holders who freeze their accounts can vote for a selection of SR candidates, with the top 27 candidates deemed the SRs. Voters may choose SRs based on criteria such as projects sponsored by SRs to

[9] PAAR, C., PELZL, J., *Understanding Cryptography: A Textbook for Students and Practitioners*, 2010 ed. Springer-Verlag Berlin Heidelberg, 2010.
[10] https://www.sciencedirect.com/science/article/pii/S2542435118301776

increase TRX adoption, and rewards distributed to voters. This allows for a more democratized and decentralized ecosystem. SRs' accounts are normal accounts, but their accumulation of votes allows them to produce blocks. With the low throughput rates of Bitcoin and Ethereum due to their PoW consensus mechanism and scalability issues, TRON's DPoS system offers an innovative mechanism resulting in 2000 TPS compared to Bitcoin's 3 TPS and Ethereum's 15 TPS.

The TRON protocol network generates one block every three seconds, with each block awarding 32 TRX to Super Representatives. A total of 336,384,000 TRX will be awarded annually to the 27 SRs. Each time an SR finishes block production, rewards are sent to a sub-account in the super-ledger. SRs can check, but not directly make use of these TRX tokens. A withdrawal can be made by each SR once every 24 hours, transferring the rewards from the sub-account to the specified SR account.

The three types of nodes on the TRON network are Witness Node, Full Node, and Solidity Node. Witness nodes are set up by SRs and are mainly responsible for block production and proposal creation/voting. Full nodes provide APIs and broadcast transactions and blocks. Solidity nodes sync blocks from other Full Nodes and also provide indexable APIs.

4. Account

4.1 Types

The three types of accounts in the TRON network are regular accounts, token accounts, and contract accounts.

1. Regular accounts are used for standard transactions.
2. Token accounts are used for storing TRC-10 tokens.
3. Contract accounts are smart contract accounts created by regular accounts and can be triggered by regular accounts as well.

4.2 Creation

There are three ways to create a TRON account:

1. Create a new account through API
2. Transfer TRX into a new account address
3. Transfer any TRC-10 token into a new account address

An offline key-pair consisting of an address (public key) and a private key, and not recorded by the TRON network, can also be generated. The user address generation algorithm consists of generating a key-pair and then extracting the public key (64-byte byte array representing x, y coordinates). Hash the public key using the SHA3-256 function (the SHA3 protocol adopted is KECCAK-256) and extract the last 20 bytes of the result. Add 41 to the beginning of the byte array and ensure the initial address length is 21 bytes. Hash the address twice using SHA3-256 function and take the first 4 bytes as verification code. Add the verification code to the end of the initial address and obtain the address in base58check format through base58 encoding. An encoded Mainnet address begins with T and is 34 bytes in length.

4.3 Structure

The three different account types are Normal, AssetIssue, and Contract. An Account contains 7 parameters:

1. **account_name**: the name for this account – e.g. BillsAccount.
2. **type**: what type of this account is – e.g. 0 (stands for type 'Normal').
3. **balance**: balance of this account – e.g. 4213312.

4. **vote**: received votes on this account – e.g. {(“0x1b7w…9xj3”,323), (“0x8djq…j12m”,88),…,(“0x82nd…mx6i”,10001)}.
5. **asset**: other assets expected TRX in this account – e.g. {<“WishToken”, 66666>, <”Dogie”, 233>}.
6. **latest_operation_time**: the latest operation time of this account.

Protobuf data structure:

```
message Account {
  message Vote {
    bytes vote_address = 1;
    int64 vote_count = 2;
  }
  bytes accout_name = 1;
  AccountType type = 2;
  bytes address = 3;
  int64 balance = 4;
  repeated Vote votes = 5;
  map<string, int64> asset = 6;
  int64 latest_operation_time = 10;
}
```

```
enum AccountType {
  Normal = 0;
  AssetIssue = 1;
  Contract = 2;
}
```

5. Block

A block typically contains a block header and several transactions.

Protobuf data structure:

```
message Block {
  BlockHeader block_header = 1;
  repeated Transaction transactions = 2;
}
```

5.1 Block Header

A block header contains **raw_data**, **witness_signature**, and **blockID**.

Protobuf data structure:

```
message BlockHeader {
  message raw {
    int64 timestamp = 1;
    bytes txTrieRoot = 2;
    bytes parentHash = 3;
    uint64 number = 4;
    uint64 version = 5;
    bytes witness_address = 6;
  }
  bytes witness_signature = 2;
  bytes blockID = 3;
}
```

5.1.1 Raw Data

Raw data is denoted as **raw_data** in Protobuf. It contains the raw data of a message, containing 6 parameters:

1. **timestamp**: timestamp of this message – e.g. 1543884429000.
2. **txTrieRoot**: the Merkle Tree's Root – e.g. 7dacsa…3ed.
3. **parentHash**: the hash of the last block – e.g. 7dacsa…3ed.
4. **number**: the block height – e.g. 4638708.
5. **version**: reserved – e.g. 5.

6. **witness_address**: the address of the witness packed in this block – e.g. 41928c...4d21.

5.1.2 Witness Signature

Witness signature is denoted as **witness_signature** in Protobuf, which is the signature for this block header from the witness node.

5.1.3 Block ID

Block ID is denoted as **blockID** in Protobuf. It contains the atomic identification of a block. A Block ID contains 2 parameters:

1. **hash**: the hash of block.
2. **number**: the hash and height of the block.

5.2 Transaction

5.2.1 Signing

TRON’s transaction signing process follows a standard ECDSA cryptographic algorithm, with a SECP256K1 selection curve. A private key is a random number, and the public key is a point on the elliptic curve. The public key generation process consists of first generating a random number as a private key, and then multiplying the base point of the elliptic curve by the private key to obtain the public key. When a transaction occurs, the transaction raw data is first converted into byte format. The raw data then undergoes SHA-256 hashing. The private key corresponding to the contract address then signs the result of the SHA256 hash. The signature result is then added to the transaction.

5.2.2 Bandwidth Model

Ordinary transactions only consume bandwidth points, but smart contract operations consume both energy and bandwidth points. There are two types of bandwidth points available. Users can gain bandwidth points from freezing TRX, while 5000 free bandwidth points are also available daily.

When a TRX transaction is broadcast, it is transmitted and stored in the form of a byte array over the network. Bandwidth Points consumed by one transaction = number of transaction bytes multiplied by bandwidth points rate. For example, if the byte array length of a transaction is 200, then the transaction consumes 200 bandwidth points. However, if a TRX or token transfer results in the target account being created, then only the bandwidth points consumed to create the account will be deducted, and additional bandwidth points will not be deducted. In an account creation scenario, the network will first consume the bandwidth points that the transaction initiator gained from freezing TRX. If this amount is insufficient, then the network consumes the transaction initiator's TRX.

In standard TRX transfer scenarios from one TRX account to another, the network first consumes the bandwidth points gained by the transaction initiator for freezing TRX. If that is insufficient, it then consumes from the free 5000 daily bandwidth points. If that is still not enough, then the network consumes the TRX of the transaction initiator. The amount is calculated by the number of bytes in the transaction multiplied by 10 SUN. Thus, for most TRX holders who may not necessarily freeze their TRX to participate in SR voting, the first step is automatically skipped (since TRX balance frozen = 0) and the 5000 daily free bandwidth powers the transaction.

For TRC-10 token transfers, the network first verifies whether the total free bandwidth points of the issued token asset are sufficient. If not, the bandwidth points obtained from freezing TRX are consumed. If there is still not enough bandwidth points, then it consumes the TRX of the transaction initiator.

5.2.3 Fee

TRON network generally does not charge fees for most transactions, however, due to system restrictions and fairness, bandwidth usage and transactions do take in certain fees.

Fee charges are broken down into the following categories:

1. Normal transactions cost bandwidth points. Users can use the free daily bandwidth points (5000) or freeze TRX to obtain more. When bandwidth points are not enough, TRX will be used directly from the sending account. The TRX needed is the number of bytes * 10 SUN.
2. Smart contracts cost energy (Section 6) but will also need bandwidth points for the transaction to be broadcasted and confirmed. The bandwidth cost is the same as above.
3. All query transactions are free. It doesn't cost energy or bandwidth.

TRON network also defines a set of fixed fees for the following transactions:

1. Creating a witness node: 9999 TRX
2. Issuing a TRC-10 token: 1024 TRX
3. Creating a new account: 0.1 TRX
4. Creating an exchange pair: 1024 TRX

5.2.4 Transaction as Proof of Stake (TaPoS)

TRON uses TaPoS to ensure the transactions all confirm the main blockchain, while making it difficult to forge counterfeit chains. In TaPoS, the networks require each transaction include part of the hash of a recent block header. This requirement prevents transactions from being replayed on forks not including the referenced block, and also signals the network that a particular user and their stake are on a specific fork. This consensus mechanism protects the network against Denial of Service, 51%, selfish mining, and double spend attacks.

5.2.5 Transaction Confirmation

A transaction is included in a future block after being broadcast to the network. After 19 blocks are mined on TRON (including its own block), the transaction is confirmed. Each block is produced by one of the top 27 Super Representatives in a round robin fashion. Each block takes ~3 seconds to be mined on the blockchain. Time may slightly vary for each Super Representative due to network conditions and machine configurations. In general, a transaction is considered fully confirmed after ~1 minute.

5.2.6 Structure

Transaction APIs consist of the following functions:

```
message Transaction {
  message Contract {
    enum ContractType {
      AccountCreateContract = 0;  // Create account/wallet
      TransferContract = 1;  // Transfer TRX
      TransferAssetContract = 2;  // Transfer TRC10 token
      VoteWitnessContract = 4;  // Vote for Super Representative (SR)
      WitnessCreateContract = 5;  // Create a new SR account
      AssetIssueContract = 6;  // Create a new TRC10 token
      WitnessUpdateContract = 8;  // Update SR information
      ParticipateAssetIssueContract = 9;  // Purchase TRC10 token
      AccountUpdateContract = 10;  // Update account/wallet information
      FreezeBalanceContract = 11;  // Freeze TRX for bandwidth or energy
      UnfreezeBalanceContract = 12;  // Unfreeze TRX
      WithdrawBalanceContract = 13;  // Withdraw SR rewards, once per day
      UnfreezeAssetContract = 14;  // Unfreeze TRC10 token
      UpdateAssetContract = 15;  // Update a TRC10 token’s information
      ProposalCreateContract = 16;  // Create a new network proposal by any SR
      ProposalApproveContract = 17;  // SR votes yes for a network proposal
      ProposalDeleteContract = 18;  // Delete a network proposal by owner
      CreateSmartContract = 30;  // Deploy a new smart contract
      TriggerSmartContract = 31;  // Call a function on a smart contract
      GetContract = 32;  // Get an existing smart contract
      UpdateSettingContract = 33;  // Update a smart contract’s parameters
      ExchangeCreateContract = 41;  // Create a token trading pair on DEX
      ExchangeInjectContract = 42;  // Inject funding into a trading pair
      ExchangeWithdrawContract = 43;  // Withdraw funding from a trading pair
      ExchangeTransactionContract = 44;  // Perform token trading
      UpdateEnergyLimitContract = 45;  // Update origin_energy_limit on a
smart contract
    }
  }
}
```

6. TRON Virtual Machine (TVM)

6.1 Introduction

TRON Virtual Machine (TVM) is a lightweight, Turing complete virtual machine developed for the TRON's ecosystem. Its goal is to provide a custom-built blockchain system that is efficient, convenient, stable, secure and scalable.

TVM initially forked from EVM[11] and can connect seamlessly with the existing solidity smart contract development ecosystem. Based on that, TVM additionally supports DPoS consensus.

TVM employs the concept of Energy. Different from the Gas mechanism on EVM, operations of transactions and smart contracts on TVM are free, with no TRX consumed. Technically, executable computation capacity on TVM is not restricted by total holding amount of tokens.

6.2 Workflow

The compiler first translates the Solidity smart contract into bytecode readable and executable on the TVM. The TVM then processes data through opcode, which is equivalent to operating the logic of a stack-based finite state machine. Finally, the TVM accesses blockchain data and invokes External Data Interface through the Interoperation layer.

[11] EVM: Ethereum Virtual Machine (https://github.com/ethereum/ethereumj)

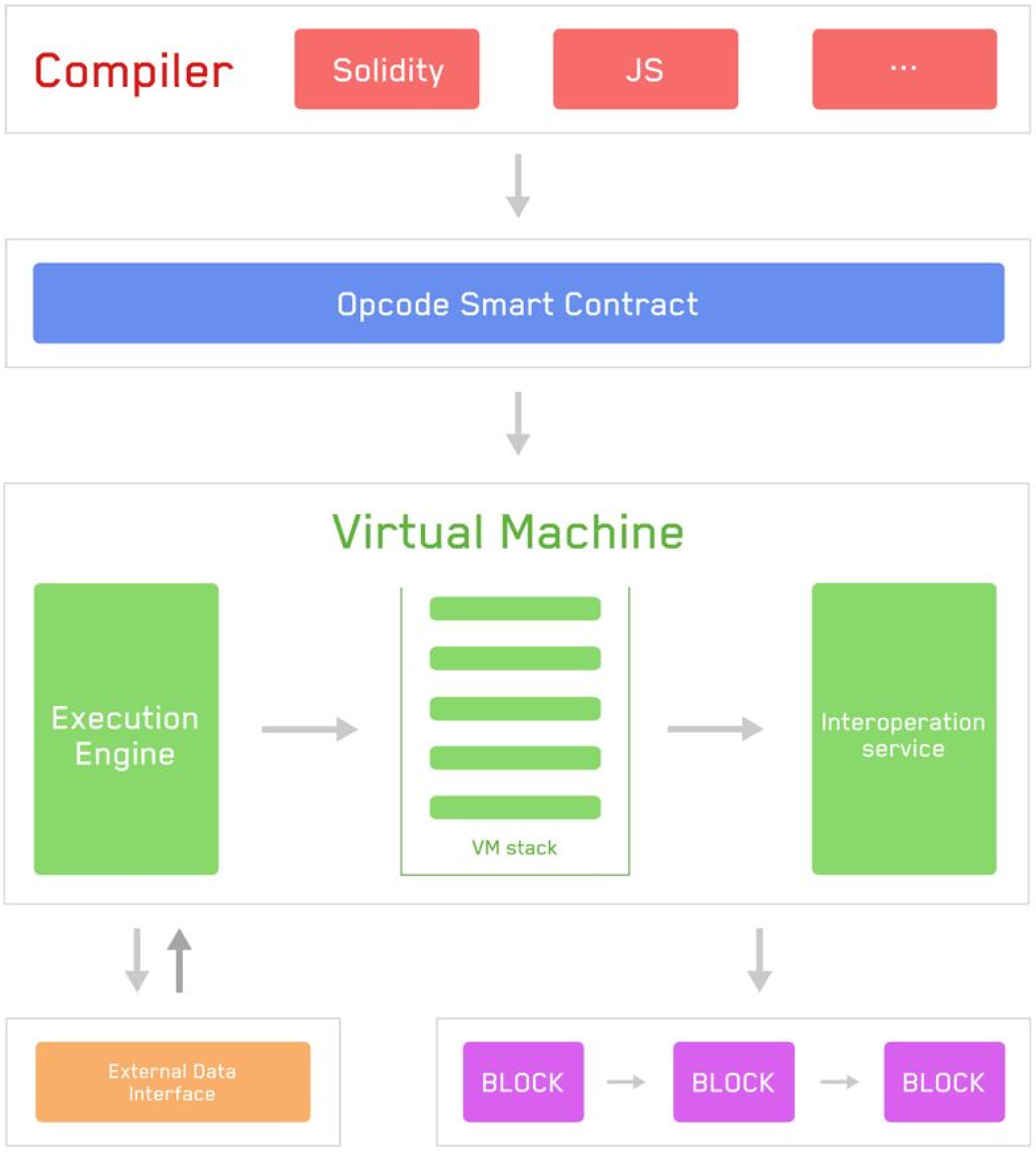


Figure 3: TVM Workflow

6.3 Performance

6.3.1 Lightweight Architecture

TVM adopts a lightweight architecture with the aim of reducing resource consumption to guarantee system performance.

6.3.2 Robust

TRX transfers and smart contract execution cost bandwidth points only, instead of TRX, which exempts TRON from being attacked. Bandwidth consumption is predictable and static since each computational step cost is fixed.

6.3.3 High Compatibility

TVM is compatible with EVM and will be compatible with more mainstream VMs in the future. Thereby, all smart contracts on EVM are executable on TVM.

6.3.4 Low Cost

Due to TVM's bandwidth setup, development costs are reduced and developers can focus on the logic development of their contract code. TVM also offers all-in-one interfaces for contract deployment, triggering and viewing to offer the convenience for developers.

7. Smart Contract

7.1 Introduction

A smart contract is a protocol that digitally verifies contract negotiation. They define the rules and penalties related to an agreement and also automatically enforce those obligations. The smart contract code facilitates, verifies, and enforces the negotiation or performance of an agreement or transaction. From a tokenization perspective, smart contracts also facilitate automatic funds transfers between participating parties should certain criteria be met.

TRON smart contracts are written in the Solidity language. Once written and tested, they can be compiled into bytecode, then deployed onto the TRON network for the TRON Virtual Machine. Once deployed, smart contracts can be queried via their contract addresses. The contract Application Binary Interface (ABI) shows the contract's call functions and is used for interacting with the network.

7.2 Energy Model

The maximum energy limit for deploying and triggering a smart contract is a function of several variables:

- Dynamic energy from freezing 1 TRX is 50,000,000,000 (Total Energy Limit) / (Total Energy Weight)
- Energy limit is the daily account energy limit from freezing TRX
- Remaining daily account energy from freezing TRX is calculated as Energy Limit - Energy Used
- Fee limit in TRX is set in smart contract deploy/trigger call
- Remaining usable TRX in the account
- Energy per TRX if purchased directly (10 SUN = 1 Energy) = 100,000, SRs can vote on adjustment

There are two consumption scenarios to calculate for maximum energy limit for deployment and trigger. The logic can be expressed as follows:

```
const R = Dynamic Energy Limit
const F = Daily account energy from freezing TRX
const E = Remaining daily account energy from freezing TRX
const L = Fee limit in TRX set in deploy/trigger call
const T = Remaining usable TRX in account
const C = Energy per TRX if purchased directly

// Calculate M, defined as maximum energy limit for deployment/trigger of
smart contract
if F > L*R
      let M = min(E+T*C, L*R)
else
      let M = E+T*C
```

7.3 Deployment

When a TRON solidity smart contract is compiled, the TRON Virtual Machine reads the compiled bytecode. The bytecode consists of a section for code deployment, contract code, and the Auxdata. The Auxdata is the source code's cryptographic fingerprint, used for verification. The deployment bytecode runs the constructor function and sets up the initial storage variables. The deployment code also calculates the contract code and returns it to the TVM. The ABI is a JSON file that describes a TRON smart contract's functions. This file defines the function names, their payability, the function return values, and their state mutability.

7.4 Trigger Function

Once the TRON smart contracts are deployed, their functions can be triggered individually either via TronStudio or through API calls. State-changing functions require Energy while read-only functions execute without Energy.

7.5 TRON Solidity

TRON Solidity is a fork from Ethereum's Solidity language. TRON modifies the original project to support TRX and SUN units (1 TRX = 1,000,000 SUN). The rest of the language syntax is compatible with Solidity ^0.4.24. Thus the Tron Virtual Machine (TVM) is almost 100% compatible with EVM instructions.

8. Token

8.1 TRC-10 Token

In the TRON network, each account can issue tokens at the expense of 1024 TRX. To issue tokens, the issuer needs to specify a token name, the total capitalization, the exchange rate to TRX, circulation duration, description, website, maximum bandwidth consumption per account, total bandwidth consumption, and the amount of token frozen. Each token issuance can also configure each account's maximum daily token transfer Bandwidth Points, the entire network's maximum daily token transfer Bandwidth Points, total token supply, locking duration in days, and the total amount of tokens locked.

8.2 TRC-20 Token

TRC-20 is a technical standard used for smart contracts implementing tokens supported by the TRON Virtual Machine. It is fully compatible with ERC-20.

The interface is as follows:

```
contract TRC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint
balance);
    function allowance(address tokenOwner, address spender) public constant
returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool
success);
    function transferFrom(address from, address to, uint tokens) public
returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint
tokens);
}
```

From a developer's perspective, there are several differences between TRC-10 and TRC-20. Some of the key differences are that TRC-10 tokens are accessible by APIs and smart contracts while TRC-20 tokens allow for interface customization but are only accessible within smart contracts.

From a cost perspective, TRC-10 tokens have transaction fees that are 1000 times lower than TRC-20, but carry bandwidth costs for API transfers and deposits. Transfers and deposits in smart contracts for TRC-10 tokens cost both bandwidth and energy.

8.3 Beyond

Since TRON uses the same Solidity version as Ethereum, more token standards could be readily ported to TRON.

9. Governance

9.1 Super Representative

9.1.1 General

Every account in the TRON network can apply and have the opportunity to become a Super Representative (denoted as SR). Everyone can vote for SR candidates. The top 27 candidates with the most votes will become SRs with the right and obligation to generate blocks. The votes are counted every 6 hours and the SRs will change accordingly.

To prevent malicious attacks, there is a cost to becoming an SR candidate. When applying, 9999 TRX will be burned from the applicant's account. Once successful, such account can join the SR election.

9.1.2 Election

TRON Power (denoted as TP) is needed to vote and the amount of TP depends on the voter's frozen assets (TRX).

TP is calculated in the following way:

$$1\ TP\ =\ 1\ TRX\ frozen\ to\ get\ bandwidth$$

Every account in the TRON network has the right to vote for their own SRs.

After the release (unfreeze, available after 3 days), users won't have any frozen assets and lose all TP accordingly. As a result, all votes become invalid for the ongoing and future voting round unless TRX is frozen again to vote.

Note that the TRON network only records the most recent vote, which means that every new vote will negate all previous votes.

9.1.3 Reward

a. Vote Reward

Also known as Candidate Reward, which the top 127 candidates updated once every round (6 hours) will share 115,200 TRX as mined. The reward will be split in accordance with the vote weight each candidate receives. Each year, the total reward for candidates will be 168,192,000 TRX.

Total vote reward per round
Why 115,200 TRX every round?

$$115,200\ TRX\ =\ total\ vote\ reward\ per\ round\ (VR/round)$$

$$VR/round\ =\ 16\ TRX/block \times 20\ blocks/min \times 60\ mins/hr \times 6\ hrs/round$$

Notice: this is set by WITNESS_STANDBY_ALLOWANCE = 115,200 TRX. See dynamic network parameters.

Total vote reward per year
Why 168,192,000 TRX every year?

$$168,192,000\ TRX\ =\ total\ vote\ reward\ per\ year\ (VR/year)$$

$$VR/year\ =\ 115,200\ TRX/round \times 4\ rounds/day \times 365\ days/year$$

b. Block Reward

Also known as Super Representative Reward, which the top 27 candidates (SRs) who are elected every round (6 hours) will share roughly 230,400 TRX as mined. The reward will be split evenly between the 27 SRs (minus the total reward blocks missed due to network error). A total of 336,384,000 TRX will be awarded annually to the 27 SRs.

Total block reward per round
Why 230,400 TRX every round?

$$230,400\ TRX\ =\ total\ block\ reward\ per\ round\ (BR/round)$$

$$BR/round\ =\ 32\ TRX/bloc \times 20\ blocks/min \times 60\ mins/hr \times 6\ hrs/round$$

Notice: the unit block reward is set by WITNESS_PAY_PER_BLOCK = 32 TRX. See dynamic network parameters.

Total block reward per year
Why 336,384,000 TRX every year?

$$336,384,000\ TRX\ = total\ block\ reward\ per\ year\ (BR/year)$$

$$BR/year\ =\ 230,400\ TRX/round \times 4\ rounds/day \times 365\ days/year$$

January 1, 2021
There will be no inflation on the TRON network before January 1, 2021, and the TRON Foundation will award all block rewards and candidate rewards prior to that date.

c. Reward Calculation

SR reward calculation

$$total\ reward\ =\ vote\ reward\ (VR)\ +\ block\ reward\ (BR)$$

$$VR\ =\ total\ VR \times \frac{votes\ SR\ candidate\ received}{total\ votes}$$

$$BR = \frac{total\ BR}{27} - block\ missed \times 32$$

Note: the reward is calculated per SR per round (6 hours)

Rank 28 to rank 127 SR candidate reward calculation

$$total\ reward\ =\ vote\ reward\ (VR)$$

$$VR\ =\ total\ VR\ \times\ \frac{votes\ SR\ candidate\ received}{total\ votes}$$

Note: the reward is calculated per SR candidate per round (6 hours)

9.2 Committee

9.2.1 General

The committee is used to modify TRON dynamic network parameters, such as block generation rewards, transaction fees, etc. The committee consists of the 27 SRs in the current round. Each SR has the right to propose and vote on proposals. When a proposal receives 19 votes or more, it is approved and the new network parameters will be applied in the next maintenance period (3 days).

9.2.2 Dynamic Network Parameters

0. MAINTENANCE_TIME_INTERVAL
    a. Description
       Modify the maintenance interval time in ms. Known as the SR vote interval time per round.
    b. Example
       [6 * 3600 * 1000] ms - which is 6 hours.
    c. Range
       [3 * 27* 1000, 24 * 3600 * 1000] ms
1. ACCOUNT_UPGRADE_COST
    a. Description
       Modify the cost of applying for SR account.
    b. Example
       [9,999,000,000] SUN - which is 9,999 TRX.
    c. Range
       [0,100 000 000 000 000 000] SUN
2. CREATE_ACCOUNT_FEE
    a. Description
       Modify the account creation fee.

   b. Example
      [100,000] SUN - which is 1 TRX.
   c. Range
      [0,100 000 000 000 000 000] SUN
3. TRANSACTION_FEE
   a. Description
      Modify the amount of fee used to gain extra bandwidth.
   b. Example
      [10] SUN/byte.
   c. Range
      [0,100 000 000 000 000 000] SUN/byte
4. ASSET_ISSUE_FEE
   a. Description
      Modify asset issuance fee.
   b. Example
      [1024,000,000] SUN - which is 1024 TRX.
   c. Range
      [0,100 000 000 000 000 000] SUN
5. WITNESS_PAY_PER_BLOCK
   a. Description
      Modify SR block generation reward. Known as unit block reward.
   b. Example
      [32,000,000] SUN - which is 32 TRX.
   c. Range
      [0,100 000 000 000 000 000] SUN
6. WITNESS_STANDBY_ALLOWANCE
   a. Description
      Modify the rewards given to the top 127 SR candidates. Known as total vote reward per round.
   b. Example
      [115,200,000,000] SUN - which is 115,200 TRX.
   c. Range
      [0,100 000 000 000 000 000] SUN
7. CREATE_NEW_ACCOUNT_FEE_IN_SYSTEM_CONTRACT
   a. Description
      Modify the cost of account creation. Combine dynamic network parameters #8 to get total account creation cost:
      $CREATE\_NEW\_ACCOUNT\_FEE\_IN\_SYSTEM\_CONTRACT \times CREATE\_NEW\_ACCOUNT\_BANDWIDTH\_RATE$
   b. Example
      [0] SUN.
   c. Range
      [0,100 000 000 000 000 000] SUN
8. CREATE_NEW_ACCOUNT_BANDWIDTH_RATE

    a. Description
       Modify the cost of account creation. Combine dynamic network parameters #7 to get total account creation cost:
       $CREATE\_NEW\_ACCOUNT\_FEE\_IN\_SYSTEM\_CONTRACT \times CREATE\_NEW\_ACCOUNT\_BANDWIDTH\_RATE$
    b. Example
       [1].
    c. Range
       [0,100,000,000,000,000,000]

9. ALLOW_CREATION_OF_CONTRACTS
    a. Description
       To turn on Tron Virtual Machine (TVM).
    b. Example
       True - set to activate and effect since 10/10/2018 23:47 UTC.
    c. Range
       True/False
10. REMOVE_THE_POWER_OF_THE_GR
    a. Description
       Remove the initial GR genesis votes
    b. Example
       True - effected at 11/4/2018 08:46 UTC.
    c. Range
       True/False - Notice: cannot set back to False from True.
11. ENERGY_FEE
    a. Description
       Modify the fee of 1 energy.
    b. Example
       20 SUN.
    c. Range
       [0,100 000 000 000 000 000] SUN
12. EXCHANGE_CREATE_FEE
    a. Description
       Modify the cost of trading pair creation. Known as the cost of creating a trade order.
    b. Example
       [1,024,000,000] SUN - which is 1024 TRX.
    c. Range
       [0,100 000 000 000 000 000] SUN
13. MAX_CPU_TIME_OF_ONE_TX
    a. Description
       Modify the maximum execution time of one transaction. Known as the timeout limit of one transaction.
    b. Example
       50 ms.
    c. Range

[0, 1000] ms

14. ALLOW_UPDATE_ACCOUNT_NAME
    a. Description
       Modify the option to let an account update their account name.
    b. Example
       False - which is available to propose from java-tron Odyssey v3.2.
    c. Range
       True/False - Notice: cannot set back to False from True.
15. ALLOW_SAME_TOKEN_NAME
    a. Description
       Modify the validation of allowing different token have a duplicate name.
    b. Example
       False - which is available to propose from java-tron Odyssey v3.2.
    c. Range
       True/False - Notice: cannot set back to False from True.
16. ALLOW_DELEGATE_RESOURCE
    a. Description
       Modify the validation of allowing to issue token with a duplicate name, so the **tokenID** of the token, in long integer data type, would be the only atomic identification of a token.
    b. Example
       False - which is available to propose from java-tron Odyssey v3.2.
    c. Range
       True/False - Notice: cannot set back to False from True.
17. TOTAL_ENERGY_LIMIT
    a. Description
       Modify the whole network total energy limit.
    b. Example
       [50,000,000,000,000,000] SUN - which is 50,000,000,000 TRX.
    c. Range
       [0,100,000,000,000,000,000] SUN
18. ALLOW_TVM_TRANSFER_TRC10
    a. Description
       Allow TRC-10 token transfer within smart contracts.
       ALLOW_UPDATE_ACCOUNT_NAME, ALLOW_SAME_TOKEN_NAME, ALLOW_DELEGATE_RESOURCE proposals must all be approved before proposing this parameter change.
    b. Example
       False - which is available to propose from java-tron Odyssey v3.2.
    c. Range
       True/False - Notice: cannot set back to False from True.

9.2.3 Create Proposal

Only the SR accounts have the rights to propose a change in dynamic network parameters.

9.2.4 Vote Proposal

Only committee members (SRs) can vote for a proposal and the member who does not vote in time will be considered as a disagree. The proposal is active for 3 days after it is created. The vote can be changed or retrieved during the 3-days voting window. Once the period ends, the proposal will either succeed (19+ votes) or fail (and end).

9.2.5 Cancel Proposal

The proposer can cancel the proposal before it becomes effective.

9.3 Structure

SRs are the witnesses of newly generated blocks. A witness contains 8 parameters:

1. **address**: the address of this witness – e.g. 0xu82h…7237.
2. **voteCount**: number of received votes on this witness – e.g. 234234.
3. **pubKey**: the public key for this witness – e.g. 0xu82h…7237.
4. **url**: the url for this witness – e.g. https://www.noonetrust.com.
5. **totalProduced**: the number of blocks this witness produced – e.g. 2434.
6. **totalMissed**: the number of blocks this witness missed – e.g. 7.
7. **latestBlockNum**: the latest height of block – e.g. 4522.
8. **isjobs**: a boolean flag.

Protobuf data structure:

```
message Witness{
  bytes address = 1;
  int64 voteCount = 2;
  bytes pubKey = 3;
  string url = 4;
  int64 totalProduced = 5;
  int64 totalMissed = 6;
  int64 latestBlockNum = 7;
  bool isJobs = 8;
}
```

10. DApp Development

10.1 APIs

The TRON network offers a wide selection of over 60+ HTTP API gateways for interacting with the network via Full and Solidity Nodes. Additionally, TronWeb is a comprehensive JavaScript library containing API functions that enable developers to deploy smart contracts, change the blockchain state, query blockchain and contract information, trade on the DEX, and much more. These API gateways can be directed towards a local privatenet, the Shasta testnet, or the TRON Mainnet.

10.2 Networks

TRON has both a Shasta testnet as well as a Mainnet. Developers may connect to the networks by deploying nodes, interacting via TronStudio, or using APIs via the TronGrid service. The TronGrid service consists of load balanced node clusters hosted on AWS servers worldwide. As DApp development scales up and API call volumes increase, TronGrid successfully fields the increase in API traffic.

10.3 Tools

TRON offers a suite of development tools for enabling developers to create innovative DApps. TronBox is a framework that allows developers to test and deploy smart contracts via the TronWeb API. TronGrid is a load balanced and hosted API service that allows developers to access the TRON network without having to run their own node. TronGrid offers access to both the Shasta testnet as well as the TRON Mainnet. TronStudio is a comprehensive Integrated Development Environment (IDE) that enables developers to compile, deploy, and debug their Solidity smart contracts. TronStudio contains an internal full node that creates a private local environment for smart contract testing prior to deployment. The TronWeb API library connects developers to the network via a wide selection of HTTP API calls wrapped in JavaScript.

10.4 Resources

The TRON Developer Hub is a comprehensive API documentation[12] site tailored towards developers wishing to build on the TRON network. The Developer Hub provides a high-level conceptual understanding of TRON and walks users through the details of interacting with the

[12] Developer Hub: https://developers.tron.network/

network. The guides walk developers through node setup, deployment and interaction with smart contracts, API interaction and implementation, building sample DApps, and using each of the developer tools. Additionally, developer community channels are available through Discord[13].

[13] Discord: https://discordapp.com/invite/GsRgsTD

11. Conclusion

TRON is a scalable blockchain solution that has employed innovative methods for tackling challenges faced by legacy blockchain networks. Having reached over 2M transactions per day, with over 700K TRX accounts, and surpassing 2000 TPS, TRON has enabled the community in creating a decentralized and democratized network.